2 Pathway ReLU - Big picture

Think of the network as doing this:

> **Build a linear operator on the fly by combining many simple "routes" (paths) through the network.**

- Each *path* contributes a very simple transformation (a rank-1 piece)
- The network output is a **sum of many such pieces**
- If you have enough independent pieces, you can build a **full-rank transformation**

---

1. What "2-pathway ReLU" changes

A standard ReLU network effectively:

- picks **one dominant path per neuron chain**
- so each region behaves like a **single rank-1 operator**

That's very restrictive.

With 2 pathways per neuron

Each neuron carries:

- a "positive" flow
- a "complementary/negative" flow

So instead of *selecting one path*, the network:

> **combines many paths at once**

Inside one input region, the network behaves like:

$$\text{Output} \approx \sum_{p} \alpha_p \, (\text{simple path operator}_p)$$

Each term is mathematically:

- an **outer product** (rank-1 matrix)

Key intuition

> **Single-path = pick one direction**
> **Two-pathway = mix many directions**

That's the whole game.

---

2. Where "span" comes in

All those simple path operators live in some set (S).

- The **span of (S)** = all linear combinations of those path operators
- That tells you **what transformations the network can build**

What you want

Inside a single region, you want:

> the combined operator to be **full rank**

That means:

- it can transform inputs in *all directions*
- no information is lost

When do you get that?

You need:

- enough active paths (roughly ≥ dimension (n))
- and those paths must be **independent**

This is where the earlier rule comes from:

Depth creates many paths
Width provides independent directions

---

3. Why first and last layers matter most

Each path contributes something like:

$$(\text{output vector}) \times (\text{input vector})^T$$

Those vectors come from:

- **last layer** → output directions
- **first layer** → input directions

Key takeaway

> **The span of all possible operators is controlled mainly by the first and last layers.**

The middle layers:

- only decide how to **combine** things (coefficients)

---

4. Enter the fast transform (FWHT)

You want the first and last layers to:

- spread information across all coordinates
- produce many independent directions
- be efficient to compute

That's exactly what the Fast Walsh–Hadamard Transform does.

What it gives you

- It's an **orthogonal transform** (like a rotation/reflection)
- It mixes all inputs together
- It's very fast: O(nlogn)

So it behaves like a **full, well-spread linear layer**

---

5. Why add random ±1 diagonals

Plain FWHT is highly structured:

- lots of symmetry
- repeated patterns

That can reduce effective diversity of paths.

The fix

Add:

- a random ±1 diagonal **before the first transform**
- a random ±1 diagonal **after the last transform**

This:

- breaks symmetry
- "randomizes" the basis
- makes directions more independent

Intuition

> FWHT = fast mixing
> random signs = de-structuring

Together:

cheap + expressive

---

6. How this fits the span / rank story

With this setup:

- First layer (random sign + FWHT)
  → gives a rich set of **input directions**
- Last layer (FWHT + random sign)
  → gives a rich set of **output directions**
- 2-pathway ReLU network in the middle
  → combines many paths

Result

Inside one region:

- you can sum many independent rank-1 pieces
- which can produce a **full-rank operator**

---

7. Backprop intuition (brief)

During training:

- gradients pass through the same transforms in reverse
- orthogonal transforms preserve information
- random signs don't destroy gradients

So:

learning still "sees" a well-mixed system

---

8. The mental model

Put it all together:

1. **Random signs + fast transform**
   spread input across many directions
2. **2-pathway ReLU layers**
   create and combine many paths
3. **Final transform + random signs**
   recombine into output space

⇒ gives a **full, flexible linear operator per region**

---

9. What to study next (math pointers)

To really understand this deeply, look into:

Linear algebra

- Rank and matrix factorization
- Outer products uv^T
- Subspace span

Randomized methods

- Random orthogonal matrices
- Johnson–Lindenstrauss lemma
- “Structured random projections”

Neural network theory

- Piecewise linear networks
- Path-based representations
- Expressivity vs rank

Transforms

- Fast Walsh–Hadamard Transform
- Orthogonal transforms and conditioning

Final takeaway

> **2-pathway ReLU turns the network into a system that builds transformations by combining many simple pieces.**
> **The FWHT + random signs ensure those pieces point in enough independent directions.**

That combination is what lets you:

- stay efficient
- but still reach **full-rank, expressive behavior inside each region**